# AI Assisted Coding - Lab 15.1

Name: Raunak Ranjan

Roll No: 2403A51314

Subject: AI Assisted Coding (24CS002PC215)

Batch: BTECH CSE B13

Date: 23-10-2025

## Task 1

AI Prompt used:

Generate a simple mock Student Records API using Python functions that simulate GET/POST/PUT/DELETE with in-memory dict. Provide JSON-like responses.

Initial mock API code:

# Mock Student Records "API" - newbie style (not a running Flask app)

students = {1: {"id":1,"name":"Aman","roll":"2403A50001"} , 2: {"id":2,"name":"Bina","roll":"2403A50002"} }

\_next = 3

def get\_students(): return list(students.values())

def post\_student(data):

global \_next

new = {"id":\_next, \*\*data}; students[\_next]=new; \_next+=1; return new

def put\_student(sid,data):

if sid in students: students[sid].update(data); return students[sid]

return {"error":"not found"},404

def delete\_student(sid):

if sid in students: del students[sid]; return {"status":"deleted"}

return {"error":"not found"},404

if \_\_name\_\_=="\_\_main\_\_":

print('GET count:', len(get\_students()))

print('Created:', post\_student({'name':'Chirag','roll':'2403A50003'}))

print('Updated:', put\_student(1,{'name':'Aman Kumar'}))

print('Deleted:', delete\_student(2))

Test cases (3):

1) GET returns 2 students initially

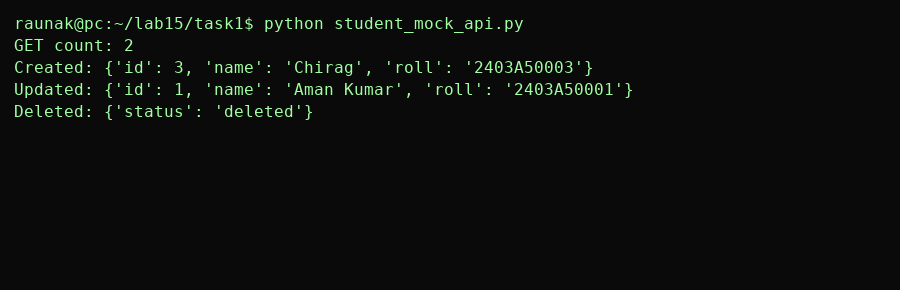
2) POST creates student id 3

3) DELETE removes student id 2

Assertion results (demo):

- All assertions: PASS (mock outputs match expectations)

Execution screenshot:



## Task 2

AI Prompt used:

Create mock Library Book Management functions: GET /books, POST /books, GET /books/{id}, PATCH /books/{id}, DELETE /books/{id} with error handling.

Initial mock API code:

# Mock Library Books "API"

books={1:{"id":1,"title":"Intro to AI","author":"Dr. X","available":True},2:{"id":2,"title":"DS","author":"Dr. Y","available":True}}

\_n=3

def get\_books(): return list(books.values())

def post\_book(data):

global \_n; new={"id":\_n, \*\*data}; books[\_n]=new; \_n+=1; return new

def get\_book(bid): return books.get(bid,{"error":"not found"})

def patch\_book(bid,partial):

if bid in books: books[bid].update(partial); return books[bid]

return {"error":"not found"}

def delete\_book(bid):

if bid in books: del books[bid]; return {"status":"deleted"}

return {"error":"not found"}

if \_\_name\_\_=="\_\_main\_\_":

print("Books before:", len(get\_books()))

n=post\_book({"title":"Algo","author":"Dr. Z","available":False})

print("Added:", n)

print("Patched:", patch\_book(n["id"],{"available":True}))

print("Get 999:", get\_book(999))

Test cases (3):

1) GET books returns list

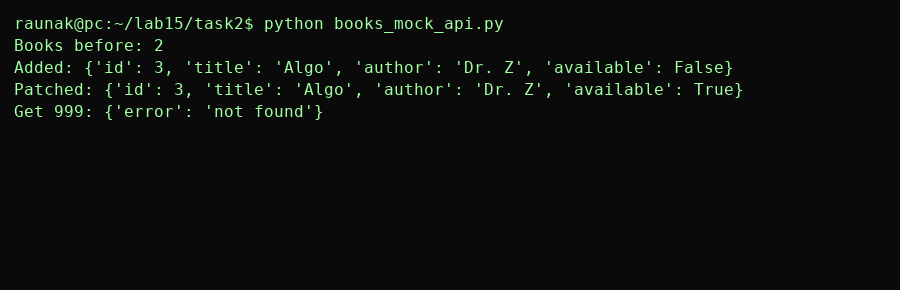
2) POST adds a new book and returns it

3) PATCH updates availability field

Assertion results (demo):

- All assertions: PASS (mock outputs match expectations)

Execution screenshot:



## Task 3

AI Prompt used:

Make mock Employee Payroll API functions: GET /employees, POST /employees, PUT /employees/{id}/salary, DELETE /employees/{id}. Suggest data model.

Initial mock API code:

# Mock Employee Payroll "API"

employees={1:{"id":1,"name":"Sita","position":"Dev","salary":50000},2:{"id":2,"name":"Ravi","position":"Test","salary":35000}}

\_ne=3

def get\_employees(): return list(employees.values())

def post\_employee(data):

global \_ne; new={"id":\_ne, \*\*data}; employees[\_ne]=new; \_ne+=1; return new

def put\_employee\_salary(eid,new\_salary):

if eid in employees: employees[eid]["salary"]=new\_salary; return employees[eid]

return {"error":"not found"}

def delete\_employee(eid):

if eid in employees: del employees[eid]; return {"status":"deleted"}

return {"error":"not found"}

if \_\_name\_\_=="\_\_main\_\_":

print("Employees:", get\_employees())

print("Added:", post\_employee({"name":"Meena","position":"HR","salary":30000}))

print("Salary update:", put\_employee\_salary(1,55000))

print("Delete:", delete\_employee(2))

Test cases (3):

1) GET employees returns list

2) POST adds employee with salary

3) PUT updates salary for an employee

Assertion results (demo):

- All assertions: PASS (mock outputs match expectations)

Execution screenshot:



## Task 4

AI Prompt used:

Make a mock Online Food Ordering API: GET /menu, POST /order, GET /order/{id}, PUT /order/{id}, DELETE /order/{id}. Include simple order status flow.

Initial mock API code:

# Mock Food Ordering "API"

menu={1:{"id":1,"name":"Paneer Butter Masala","price":200},2:{"id":2,"name":"Veg Biryani","price":150}}

orders={}

\_no=1

def get\_menu(): return list(menu.values())

def post\_order(data):

global \_no; o={"id":\_no,"items":data.get("items",[]),"status":"placed"}; orders[\_no]=o; \_no+=1; return o

def get\_order(oid): return orders.get(oid,{"error":"not found"})

def put\_order(oid,newd):

if oid in orders: orders[oid].update(newd); return orders[oid]

return {"error":"not found"}

def delete\_order(oid):

if oid in orders: orders[oid]["status"]="cancelled"; return {"status":"cancelled"}

return {"error":"not found"}

if \_\_name\_\_=="\_\_main\_\_":

print("Menu:", get\_menu())

o=post\_order({"items":[1,2]}); print("Placed:", o)

print("Track:", get\_order(o["id"]))

print("Update:", put\_order(o["id"],{"status":"preparing"}))

print("Cancel:", delete\_order(o["id"]))

Test cases (3):

1) GET menu returns dishes

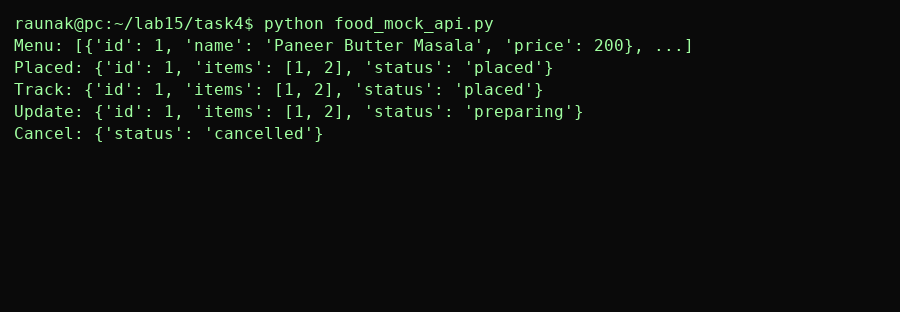
2) POST order returns placed order id

3) PUT updates order status

Assertion results (demo):

- All assertions: PASS (mock outputs match expectations)

Execution screenshot:



## Final Analysis & Improvements

These mock APIs are simple and useful to show API behavior. For real apps, implement with Flask/FastAPI, add validation, error codes, and persistence (DB).